



TITLE:

# A Formal Semantics for Algorithmic Languages Based on the Scott's Logic (アルゴリズムにおける証明論)

AUTHOR(S):

MUKAI, KUNIAKI; SEKIMOTO, SHOJI; SUDO, MASARU

---

CITATION:

MUKAI, KUNIAKI ...[et al]. A Formal Semantics for Algorithmic Languages Based on the Scott's Logic (アルゴリズムにおける証明論). 数理解析研究所講究録 1975, 236: 38-45

ISSUE DATE:

1975-05

URL:

<http://hdl.handle.net/2433/105507>

RIGHT:

A Formal Semantics For Algorithmic LanguagesBased On The Scott's Logic

K. Mukai, S. Sekimoto, and M. Sudo

(Mitsubishi Electric Corp.)

Abstract A semantics of algorithmic languages are given applying the least fixed point theory. The language to be studied in this paper is a subset of ALGOL60, which is almost the full set of ALGOL60, but without goto statements.

Mainly, meanings of mutually defined recursive procedure with parameters are described. A lattice theoretical foundation for this semantics is seen in the appendix.

§1 Introduction

In Scott<sup>[1], [3]</sup> and Strachey<sup>[1]</sup>, mathematical except (7) and (8) domains are introduced to describe the meanings of the source language. The list of those is (1) T: truth values (2) N: numbers (3) S: states (4) L: locations (5) P: procedures (6) C: commands (7) Id: identifiers (8) Env: environments.

These domains satisfy the next relations. a)  $V = T + N + C + P + L$  b)  $P = [V \rightarrow [S \rightarrow V \times S]]$  c)  $C = [S \rightarrow S]$  d)  $Env = [Id \rightarrow C + V]$ . Here  $X + Y$  means direct sum of continuous lattices  $X$  and  $Y$ , and also  $X + Y$  becomes a continuous lattice.  $[X \rightarrow Y]$  denotes the set of all continuous functions from  $X$  to  $Y$ .

The syntax of the source language studied in the above papers is divided into two categories, that is, COMMANDS (Cmd) and EXPRESSIONS (Exp). And then, two functions  $\mathcal{C}$  and  $\mathcal{E}$  are constructed recursively.

$$(1) \mathcal{C} : \text{Cmd} \rightarrow [\text{Env} \rightarrow [S \rightarrow S]]$$

$$(2) \mathcal{E} : \text{Exp} \rightarrow [\text{Env} \rightarrow [S \rightarrow V \times S]]$$

Meanings of  $\mathcal{C}$  in Cmd and  $\mathcal{E}$  in Exp are  $\mathcal{C}[[c]]$  and  $\mathcal{E}[[e]]$  respectively. Detailed explanations of Cmd, Exp,  $\mathcal{C}$ , and  $\mathcal{E}$  are, however, omitted here. (See [1] or [3]).

## §2 Recursive Procedures and Block Structures

The source language  $\mathcal{A}$ , whose semantics we are going to study, is the largest subset of ALGOL60 such that  $\mathcal{A}$  is without (1) goto statements and labels, (2) for statements, and (3) switch, own, and array.

### 2.1 Procedures

We have to modify b) in §1 into b') because in the language  $\mathcal{A}$ , a number of parameters of a procedure may be taken to be arbitrarily finite, and procedures are not restricted to function type. That is, also command type procedures are allowed.

$$b') \quad P = c + [S \rightarrow V \times S] + [V \rightarrow P]$$

### 2.2 Call by name and call by value

The language  $\mathcal{A}$  allows a flexible parameter passing i.e. call by name and call by value. The format of a procedure declaration is:

$$\langle \text{identifier} \rangle (a, b, \dots, c : x, y, \dots, z) \langle \text{procedure body} \rangle ,$$

where  $a, b, \dots, c$  and  $x, y, \dots, z$  are formal parameters called by value and name respectively. And the format of a procedure statement is:

$$\text{Call } \langle \text{identifier} \rangle (e_1, \dots, e_m : f_1, \dots, f_n) \text{ ----- } (\alpha)$$

### 2.3 A semantics of procedure statements

We describe a semantics of a procedure statement a step by step. For a given  $\sigma$  in  $S$  and  $\rho$  in  $Env$ ,

$$(1) \mathcal{E}[\mathbf{e}](\rho)(\sigma) = \langle \nu_1, \sigma_1 \rangle$$

$$\mathcal{E}[e_m](\rho)(\sigma_{m-1}) = \langle \nu_m, \sigma_m \rangle$$

$$(2) \rho' = \rho[\nu_1/a, \dots, \nu_m/c, \mathcal{E}[f_1](\rho)/x, \dots, \mathcal{E}[f_n](\rho)/z]$$

$$(3) \text{ finally } \sigma' = \mathcal{C}[\llbracket \text{procedure body} \rrbracket](\rho')(\sigma_m)$$

def. 1  $\mathcal{C}[\llbracket \alpha \rrbracket](\rho)(\sigma) \triangleq \sigma'$

We have to suppose that  $\rho$  has two facilities: briefly,

(i) a facility of "copy rule" for identifiers in a given source program in  $\mathcal{A}$ , and (ii) a dynamic memory allocation without destructing any necessary area.

### 2.4 A semantics of function procedures

For simplicity, we consider a function procedure, whose formal parameters are only  $a$  and  $x$ , and they are called by value and name respectively. We describe a meaning of next  $(\beta)$  as in section 2.3.

$$\mathcal{X}(e, f) \quad (\mathcal{X} \text{ in Id}) \quad \text{-----} \quad (\beta)$$

For given  $\rho$  in  $Env$  and  $\sigma$  is  $S$ ,

$$(1) \mathcal{E}[e](\rho)(\sigma) = \langle \nu_1, \sigma_1 \rangle$$

$$(2) \rho' = \rho[\nu_1/a, \mathcal{E}[f](\rho)/x]$$

$$(3) \sigma' = \mathcal{C}[\llbracket \text{procedure body} \rrbracket](\rho')(\sigma)$$

$$(4) \text{ Contents } (\rho(\mathcal{X}), \sigma') = \nu', \text{ (see [1] or [3]), so we obtain}$$

Def. 2  $\mathcal{E}[\llbracket \beta \rrbracket](\rho)(\sigma) \triangleq \langle \nu', \sigma' \rangle$

## 2.5 Block structure

Blocks are considered to be procedures with no parameters.

For a source program  $\pi$  in  $\mathcal{A}$ , let  $B(\pi)$  be the set of all occurrences of blocks in  $\pi$ . Each occurrence of a block in  $\pi$  is implicitly supposed to have an identifier in  $\text{Id}$ . That is,  $B(\pi) \hookrightarrow \text{Id}$  (inclusion map is, say,  $i_\pi$ ). Then for each  $b$  in  $B(\pi)$  and  $\rho$  in  $\text{Env}$ ,  $\rho(i_\pi(b))$ , or in short,  $f(b)$ , is in  $C+V$ .

It may be more useful to introduce a few notions before we finally describe a meaning of  $\pi$ . Let  $D$  be a continuous lattice.

def. 3  $\mathcal{T}(D)$  is the set of all trees over  $D$ .

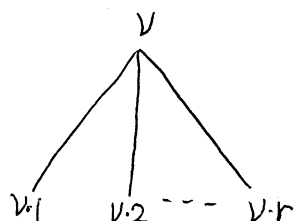
def. 4 For  $s$  and  $t$  in  $\mathcal{T}(D)$ ,  $s \leq t$  is true if and only if  $\text{dom}(s) = \text{dom}(t)$ , and  $s(\nu) \sqsubseteq t(\nu)$  for all  $\nu$  in  $\text{dom}(s)$ , where  $\sqsubseteq$  is the order relation in  $D$ .

def. 5 Let  $F = \bigcup_{n=0}^{\infty} [D^n \rightarrow D]$ , and each element in  $F$  is supposed to be assigned a rank in the apparent way.

Hereafter, a tree  $f$  over the ranked set  $F$  is fixed.

def. 6  $\mathcal{T}(D, f) \triangleq \{t \in \mathcal{T}(D) \mid \text{dom}(t) = \text{dom}(f)\}$

def. 7 A transformation  $\mathfrak{A}(f)$  on  $\mathcal{T}(D, f)$  is defined as follows,



i.e.  $t' = \mathfrak{A}(f)(t)$  is true if and only if  $t'(\nu) = f(\nu)(t(\nu.1), \dots, t(\nu.r))$  for each  $\nu$  in  $\text{dom}(f)$ . (A tree is a function from a tree domain to a set of labels.)

def. 8 By  $\text{Fix}(\mathfrak{A}(f))$ , the least fixed point of above  $\mathfrak{A}(f)$  is denoted.

## 2.6 A semantics of a source program $\pi$ in $\mathcal{A}$

In this section, we give, finally, a definition of a meaning of a source rprogram  $\pi$  in  $\mathcal{A}$ .

(1) First, transform  $\pi$  into a tree  $f$  over  $F$  precisely preserving the block structure of  $\pi$ , where  $F = \bigcup_{n=0}^{\infty} [C^n \rightarrow C]$ , and  $(C = [S \rightarrow S])$ .

For given  $\rho$  in Env, it is not so difficult to construct a tree  $f(=f_{\pi, \rho})$  mentioned above. For example, let's consider a block (#).

(#) begin if  $\epsilon$  then begin A end else B end end

The rule corresponding to (#) is

$\lambda(u, v) \text{ Cond}(u, v) * \mathcal{E}[\epsilon](\rho)$  in  $[C \times C \rightarrow C]$ . We suppose a construction of such  $f_{\pi, \rho}$  ( $=f$  for brief)

(2) Second, now we are able to get  $\mathcal{J}(c, f)$ ,  $\mathcal{A}(f)$ , and  $\text{Fix}(\mathcal{A}(f))$  as in def. 6, def. 7, and def. 8 respectively. Let  $\theta_0$  in  $C$  be the label of the root of  $\text{Fix}(\mathcal{A}(f))$ . And, at last,  $C[\pi](\rho)$  is defined to be  $\theta_0$ .

def. 9  $C[\pi](\rho) \triangleq \theta_0$

## §3 Conclusion

The method described in §2 is an extension of one used in such an example as follows.

example:

$\text{FACT}(x) := \text{if } x=0 \text{ then } 1 \text{ else } x * \text{FACT}(x-1)$

The least solution of this example is given to be  $\lim_{n \rightarrow \infty} f^{(n)}$ , where  $f^{(0)} \triangleq$  totally undefined function, and  $f^{(n+1)}(x) \triangleq$  if  $x=0$  then else  $x * f^{(n)}(x-1)$  for  $n \geq 1$ .

Without an assumption of memory (or state vectors) of a machine and a stack manipulation, it is, we think, almost impossible to describe a meaning of mutually recursive procedures together with assignment statements. And a detailed description of  $\int$  in Env corresponds to the stack manipulation or the dynamic memory allocation.

One of motivations of this essay is to give a formal description of a source program in  $\mathcal{A}$  in a bit more syntax-directed method. The def. 7 in 2.5 reminds us of D. Knuth's semantics theory for context-free languages. We have given in this paper only an outline of semantics of  $\mathcal{A}$ . More detailed discussions must be done.

#### Acknowledgements

We are grateful both to Prof. T. Kasami of the Osaka Univ. and Dr. K. Torii at ETL., for their helpful advices to our study.

#### Reference

- (1) D. Scott and C. Strachey "Toward A Mathematical Semantics For Computer Languages." (Computers and Automata, Brooklyn Inst.
- (2) D. Scott "Continuous Lattices" (Proc., Dalhousie Conf., Springer Lecture Note)
- (3) D. Scott "Mathematical Concept in Programming Languages" (SJCC, 1972)
- (4) D. Scott "The Lattices of Flow Diagrams" (Springer Lecture Notes in Mathematics 188, 311-366 (E. Engeler, Ed., 1971))
- (5) P. Naur (Ed.) "Report on the algorithmic languages ALGOL60" CACM3 (1960)

Appendix.

We show in this appendix an outline of a proof for a proposition on which our work is mainly based. From here on, a continuous lattice is restricted to such a space whose greatest element is isolated in the sense of topology.

Proposition A. For given continuous lattices  $P_0$ ,  $V_0$ , and  $S$  there exist continuous lattices  $P$  and  $V$  which satisfy the relations a1) and a2):

$$a1) \quad V \cong V_0 + P,$$

$$a2) \quad P \cong P_0 + [S \rightarrow V \times S] + [V \rightarrow P].$$

Here  $X \cong Y$  means that there exists some one-to-one and onto correspondence between continuous lattices  $X$  and  $Y$  under which  $X$  and  $Y$  are both isomorphic and homeomorphic in the sense of lattice and topological space, respectively.

The proposition A is a result of a series of lemmas listed below.

Lemma 1. i) If  $X$  and  $Y$  are continuous lattices, then  $X+Y$ ,  $X \times Y$ , and  $[X \rightarrow Y]$  are also continuous lattices.

ii) If  $\langle X_n, j_n \rangle_{n \geq 0}$  is an inverse system of continuous lattices such that for each  $n \geq 0$   $j_n$  is a projection, then the inverse limit  $\varprojlim_{n \geq 0} \langle X_n, j_n \rangle$  is also a continuous lattice.

Lemma 2. If  $X$  and  $Y$  are continuous lattices, then both  $X$  and  $Y$  are projections of  $X+Y$ .

Lemma 3. If  $X_1$ ,  $X_2$ ,  $Y_1$ , and  $Y_2$  are continuous lattices, and  $X_1$  and  $X_2$  are projections of  $Y_1$  and  $Y_2$  respectively, then next holds.



- i)  $X_1 + X_2$  is a projection of  $Y_1 + Y_2$ ,
- ii)  $X_1 \times X_2$  is a projection of  $Y_1 \times Y_2$ ,
- iii)  $[X_1 \rightarrow X_2]$  is a projection of  $[Y_1 \rightarrow Y_2]$ .

Lemma 4. If  $\langle X_n, j_n \rangle_{n \geq 0}$  and  $\langle Y_n, k_n \rangle_{n \geq 0}$  are inverse systems such that for each  $n \geq 0$ , both  $j_n$  and  $k_n$  are projections, then next holds.

- i)  $\varprojlim_{n \geq 0} \langle X_n + Y_n, j_n + k_n \rangle \cong \varprojlim_{n \geq 0} \langle X_n, j_n \rangle + \varprojlim_{n \geq 0} \langle Y_n, k_n \rangle$ ,
- ii)  $\varprojlim_{n \geq 0} \langle X_n \times Y_n, j_n \times k_n \rangle \cong \varprojlim_{n \geq 0} \langle X_n, j_n \rangle \times \varprojlim_{n \geq 0} \langle Y_n, k_n \rangle$ ,
- iii)  $\varprojlim_{n \geq 0} \langle [X_n \rightarrow Y_n], j_n \rightarrow k_n \rangle \cong \left[ \varprojlim_{n \geq 0} \langle X_n, j_n \rangle \rightarrow \varprojlim_{n \geq 0} \langle Y_n, k_n \rangle \right]$ .

Here, for each  $n \geq 0$   $j_n + k_n$  is the canonical projection from  $X_{n+1} + Y_{n+1}$  to  $X_n + Y_n$  derived from  $j_n$  and  $k_n$ , and similar is the case with either  $j_n \times k_n$  or  $j_n \rightarrow k_n$ .

(An outline of a proof for the proposition A).

First, construct two inverse systems  $\langle P_n, j_n \rangle_{n \geq 0}$  and  $\langle V_n, k_n \rangle_{n \geq 0}$ . This is all right, i.e.,  $P_n$  ( $n \geq 0$ ) is defined by an inductive method as follows.

$$P_{n+1} = P_0 + [S \rightarrow V_n \times S] + [V_n \rightarrow P_n],$$

$$V_{n+1} = V_0 + P_n.$$

Of course, using lemma 1, 2, and 3, for each  $n \geq 0$  both  $j_n$  and  $k_n$  can be taken as projections.

Second, the pair of  $P = \varprojlim_{n \geq 0} \langle P_n, j_n \rangle$  and  $V = \varprojlim_{n \geq 0} \langle V_n, k_n \rangle$  is easily checked to satisfy the relations a1) and a2) by applying the lemma 4.